Hybrid Analytical-Statistical Modeling for Efficiently Exploring Architecture and Workload Design Spaces

Lieven Eeckhout Koen De Bosschere

Department of Electronics and Information Systems (ELIS)
Ghent University, Belgium
E-mail: {leeckhou,kdb}@elis.rug.ac.be

Abstract

Microprocessor design time and effort are getting impractical due to the huge number of simulations that need to be done to evaluate various processor configurations for various workloads. An early design stage methodology could be useful to efficiently cull huge design spaces to identify regions of interest to be further explored using more accurate simulations. In this paper, we present an early design stage method that bridges the gap between analytical and statistical modeling. The hybrid analytical-statistical method presented here is based on the observation that register traffic characteristics exhibit power law properties which allows us to fully characterize a workload with just a few parameters which is much more efficient than the collection of distributions that need to be specified in classical statistical modeling. We evaluate the applicability and the usefulness of this hybrid analytical-statistical modeling technique to efficiently and accurately cull huge architectural design spaces. In addition, we demonstrate that this hybrid analytical-statistical modeling technique can be used to explore the entire workload space by varying just a few workload parameters.

1. Introduction

The efforts to design future microprocessors are ever increasing as microprocessor designs and workloads are becoming more and more complex. This is reflected in the huge number of simulations that need to be run in order to obtain a cost-effective and high-performance design. Indeed, numerous processor configurations need to be evaluated for various workloads, which results in long simulation times. It can be expected that exploring huge design spaces even through high-level architectural simulations will become impractical if not impossible in the near future given time-to-market considerations [1]. So, there is a need for methods to efficiently cull huge design spaces in early design stages. These early design stage methods should identify regions of interest that could be further explored in more detail by more accurate architectural simulations.

In this paper, we present an early design stage methodology that bridges the gap between analytical and statistical modeling. This *hybrid analytical-statistical* modeling technique is based on the observation that register traffic characteristics exhibit *power law* properties. E.g., the distribution of the age of register instances, or the number of dynamic instructions between writing and reading the same register, has a power law probability distribution function $P\left[X = x\right] = \alpha x^{-\beta}$. This previously unknown property allows us to characterize register traffic by means of two parameters, α and β, which is much more efficient than the collection of distributions that is used in the classical statistical modeling methodology [2, 4, 5, 12, 13]. The resulting hybrid analytical-statistical workload model provides opportunities to explore the *entire* workload space by varying the analytical workload parameters, namely α and β. These workload parameters can be varied to model future workloads and in addition, the interaction between architectural and workload parameters can be investigated freely which is impractical, if not impossible, using real benchmarks since the latter are just isolated points in the design space. Next to demonstrating that the hybrid analytical-statistical method can be used to explore the workload space, we also show that hybrid analytical-statistical modeling can be used to perform architecture design space explorations in an efficient and accurate manner.

This paper is organized as follows. In section 2, we present the benchmarks that we have used in this study. In section 3, we identify register traffic characteristics and we show its power law properties. In section 4, we show how distribution fitting was done to estimate parameters that characterize the register traffic in applications. Section 5 presents the hybrid analytical-statistical methodology which is shown to be useful to perform architecture and workload space explorations in sections 6 and 7, respec-

tively. Finally, we end with related work and conclusions.

2. Benchmarks

To increase the validity of the observed register traffic characteristics, we included two different workloads, namely SPECint95 and IBS [15], see Tables 1 and 2. These two workloads are two different types of workload and were obtained using different compilers for different instruction set architectures. The SPECint95 traces were generated on a DEC 500au station with an Alpha 21164 processor. The Alpha architecture is a load/store architecture and has 32 integer and 32 floating-point registers, each of which is 64 bits wide. The SPECint95 benchmarks have been compiled with the DEC cc compiler version 5.6 with the optimization flag set to -O4 and linked statically using the -non_shared flag. The traces were carefully selected not to include initialization code. The IBS traces were generated on a MIPS-based DEC 3100 system running the Mach 3.0 operating system. We included the IBS traces in our study because these traces are known to have a larger instruction footprint (due to the inclusion of significant amounts of operating system activity) and to stress the memory subsytem more than SPECint benchmarks do [15]. We excluded the zero register r0 from all measurement.

3. Register Traffic Characteristics

In modern out-of-order architectures, the registers are the primary means for inter-operation communication. When an operation writes a value into a register, a new *register instance* is *created*. Operations that read a register, are said to *use* the corresponding register instance. Franklin and Sohi [6] proposed four metrics to characterize the register traffic in modern out-of-order architectures. The first metric is the *degree of use of register instances* which measures the number of times a register instance is used by other operations. The three remaining metrics quantify the *temporal locality of creation and use of register instances*: (i) the *age of register instances*, i.e., the number of (dynamic) operations between the use and the creation of a register instance; (ii) the *useful lifetime of register instances*, i.e., the number of operations between the creation and the last use of a register instance; and (iii) the *lifetime of register instances*, i.e., the number of operations between two consecutive writes to the same register.

We have measured these four metrics for the benchmarks mentioned in the previous section. The probability distribution functions (PDFs) shown in Figure 1 are averaged over the IBS traces. These PDFs clearly follow a straight line when displayed in a log-log diagram. A PDF that follows a straight line in a log-log diagram is called a *power law* distribution or a *Pareto* distribution with probability distribution function $P\left[X = x\right] = \alpha x^{-\beta}$ for which $1 > \alpha > 0$ is the intersect of the PDF with the Y axis and $\beta > 0$ is the slope of the straight line of the PDF in a log-log diagram.

We observed these power law properties for the individual IBS traces as well as for most other benchmarks. The benchmarks for which we did not observe power law properties are benchmarks with small instruction footprints that spend most of their time in small loops, e.g., li and compress, see Tables 1 and 2 where the number of static instructions is shown that account for 90% of the dynamic instruction count. The log-log diagram for these programs drops off more quickly for higher values on the X axis. We feel that this does not affect the conclusion that the observed register traffic characteristics exhibit power law properties. This conclusion is motivated by the fact that real workloads—which should be of interest to computer architects since they build systems for real workloads—are known to have larger instruction footprints than some SPECint benchmarks and will thus exhibit similar register traffic characteristics as the benchmarks with large footprints presented here. As far as we know, we are the first to report that these register traffic characteristics show power law properties.

It is interesting to note that a power law distribution with PDF $P\left[X = x\right] = \alpha x^{-\beta}$ has an infinite mean when $\beta \leq 2$. The slopes of the age, the useful lifetime and the lifetime of register instances PDFs are ≤ 2, e.g., the slope of the age PDF is 1.59. As a result, the significance of the average age of register operands (also called the average dependency distance between operations by some authors [8, 13]) is limited.

4. Distribution Fitting

As said in the introduction, we will use the fact that the PDF of the age of register instances exhibits power law properties to obtain an analytical workload model that will be used in a hybrid analytical-statistical modeling environment to estimate microprocessor performance. To obtain an abstract workload model that is based on a small number of parameters, we have to approximate measured distribution functions with theoretical distribution functions by estimating parameters, also called distribution fitting. In the analytical workload model, the parameters of the theoretical distribution then describe the workload. To estimate theoretical distribution parameters concerning the age of register instances we take the following approach. The PDF of the age of register instances $P\left[X = x\right]$ can be written as

$$P\left[X = x\right] = P\left[X = x \mid X \geq x\right] \cdot P\left[X \geq x\right],\ x \geq 1$$

where $P\left[X = x \mid X \geq x\right]$ could be defined as the *conditional dependence probability* $1 - p_x$ (p_x corresponds to

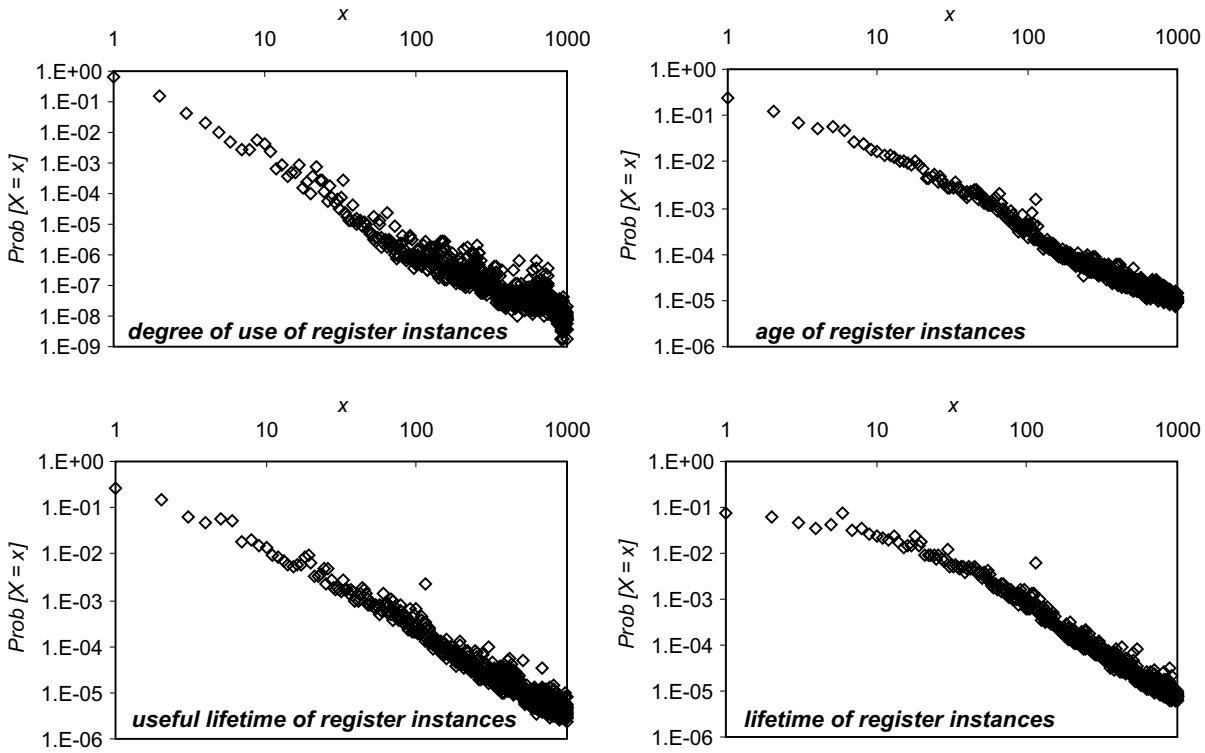

Figure 1. Register traffic metrics measured for the IBS traces: (i) degree of use, (ii) age, (iii) useful lifetime and (iv) lifetime of register instances.

the *conditional independence probability* defined by Dubey, Adams and Flynn [3]); i.e., p_x is the probability that an operation is independent on an instruction that comes x operations ahead in the instruction trace given that the operation is independent of the $x-1$ operations ahead of that operation. Using induction it can be easily verified that $P[X=x]$ can be written as

$$P\left[X=x\right] = (1-p_x)\cdot(1-\sum_{i=1}^{x-1} P\left[X=i\right])$$

$$= (1-p_x)\cdot\prod_{i=1}^{x-1} p_i,\ x \geq 1.$$

Calculating p_x from the measured $P\left[X=x\right]$ can be done as follows:

$$p_x = 1 - \frac{P\left[X=x\right]}{1-\sum_{i=1}^{x-1} P\left[X=i\right]}.$$

Kamin, Adams and Dubey [9] approximated the conditional independence probability p_x by an exponential function

$$p_x \approx 1-\alpha e^{-\beta x},$$

where α and β are constants that are determined through regression techniques applied to the measured p_x.

Our data on the other hand, reveal that p_x can be more accurately approximated by a power law function

$$p_x \approx 1-\alpha x^{-\beta}.$$

This is shown for the IBS trace real_gcc in Figure 2.

The distribution fitting was done by minimizing the sum of squared errors between the theoretical distributions and the measured data of the conditional dependence probability. This minimization gives a higher weight to smaller values of x. This choice is motivated by the fact that we want to use this approximation as an abstract workload model for a hybrid analytical-statistical performance modeling technique in which an accurate approximation of the measured data for small values of x is necessary to obtain accurate performance predictions. As can be seen from Figure 2 for real_gcc, the power law approximation is much more accurate than the exponential approximation proposed by Kamin, Adams and Dubey [9]; we experimentally verified that this also leads to a higher performance accuracy. For most benchmarks, we observed a comparable approximation quality as in Figure 2. For some benchmarks however, the exponential approximation is more accurate; this is

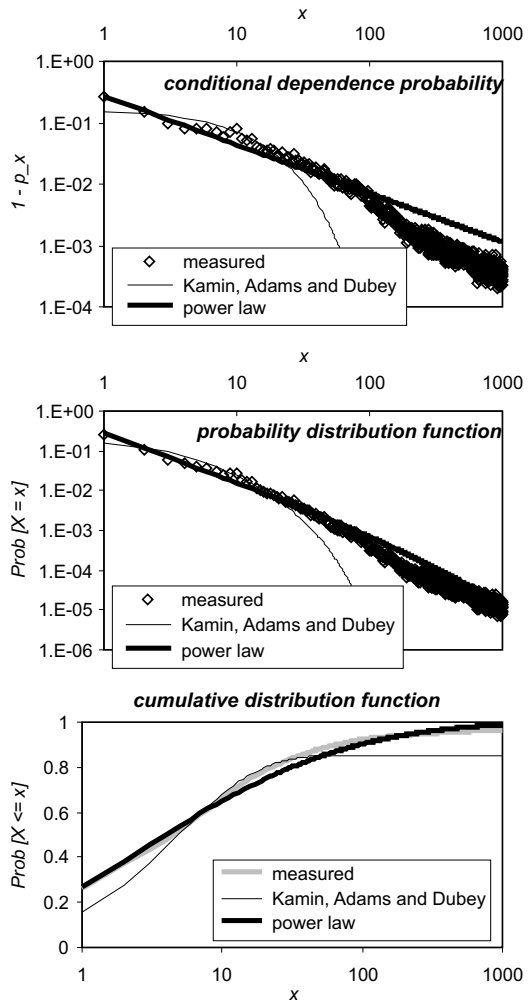

Figure 2. Parameter estimation of the conditional dependence probability p_x of the age of register instances for real_gcc**. The corresponding probability and cumulative distribution functions are as well.**

true for benchmarks with small instruction footprints, e.g., li and compress. As said before, this does not keep us from concluding that the power law approximation outperforms the exponential approximation since computer architects are mainly interested in real workloads when designing real microprocessors.

We also experimented with fitting theoretical distributions to the measured PDF data instead of the conditional dependence probability. We found that fitting to the conditional dependence probability generally yields a more accurate approximation, which motivates our approach.

The α and β values obtained from distribution fitting are listed in Tables 1 and 2 for the various benchmarks: α varies between 0.20 and 0.33 and β varies between 0.60 and 0.98.

5. Hybrid Analytical-Statistical Modeling

We first describe the 'classical' statistical simulation technique after which we show what needs to be done to obtain a hybrid analytical-statistical modeling methodology.

5.1. Statistical Modeling

The statistical simulation methodology works in three steps: statistical profiling, synthetic trace generation and trace-driven simulation.

Statistical Profiling. During the statistical profiling step, a real program trace is analyzed by a *microarchitecture-dependent* profiling tool and a *microarchitecture-independent* profiling tool. The complete set of statistics collected during statistical profiling is called a *statistical profile*. The microarchitecture-independent profiling tool extracts statistics concerning the instruction mix (we identify 19 instruction classes according to semantics and number of source registers), the age of the input register instances (measured per instruction class and per source register, 22 distributions in total) and the age of memory instances.

The microarchitecture-dependent profiling tool extracts statistics concerning the branch and cache behaviour of the program trace for a specific branch predictor and a specific cache organization. The *branch statistics* consist of seven probabilities: (i) the conditional branch target prediction accuracy, (ii) the conditional branch (taken/not-taken) prediction accuracy, (iii) the relative branch target prediction accuracy, (iv) the relative call target prediction accuracy, (v) the indirect jump target prediction accuracy, (vi) the indirect call target prediction accuracy and (vii) the return target prediction accuracy. The reason to distinguish between these seven probabilities is that the prediction accuracies greatly vary among the various branch classes. In addition, the penalties introduced by these are completely different. A misprediction in cases (i), (iii) and (iv) only introduces a single-cycle bubble in the pipeline. Cases (ii), (v), (vi) and (vii) on the other hand, will cause the entire processor pipeline to be flushed and to be refilled when the mispredicted branch is executed.

The *cache statistics* include two sets of distributions: the D-cache and the I-cache statistics. The D-cache statistics contain two probabilities, namely the probability that a load operation needs to access the L2 cache (L1 cache miss) and main memory (L2 cache miss) to get its data. The I-cache statistics consists of two probabilities as well, namely the probability that the fetch unit needs to access the L2 cache and main memory to get an instruction.

A statistical profile can be computed from an actual trace but it is more convenient to compute it on-the-fly from either an instrumented functional simulator, or from an instrumented version of the benchmark program running on a real system which eliminates the need to store huge traces. A second note is that although computing a statistical profile might take a long time, it should be done only once for each benchmark with a given input. And since statistical simulation is a fast analysis technique, computing a statistical profile will be worthwhile. A third important note is that making a distinction between microarchitecture-dependent and -independent characteristics implies that statistical simulation cannot be used to study branch predictors or cache organizations; other microarchitectural parameters however, like window size, issue width, instruction latency, etc. can be varied freely. A fourth note to be made is that statistical simulation makes no distinction between different program phases. If this would ever limit the accuracy of statistical simulation, a program trace could be easily segmented and different profiles could be used for each program phase.

Synthetic Trace Generation. Once a statistical profile is computed, a *synthetic trace* is generated by a synthetic trace generator using this statistical profile. This is done à la Monte Carlo: a random number is generated, that will determine a program characteristic using a cumulative distribution function.

The generation of a synthetic trace itself works on an operation-by-operation basis. Consider the generation of operation x in the synthetic instruction stream:

1. Determine the instruction type using the instruction-mix distribution.
2. For each source operand, determine the operation that creates this register instance using the age of register instances distribution. Notice that when a dependency is created in this step, we cannot assure that the operation that is the creator of that register instance, is neither a store nor a conditional branch operation[1]. The demand of syntactical correctness does not allow us for assigning a destination operand to a store and a conditional branch instruction. This problem is solved as follows: look for another creator instruction until we get one that is not a store nor a conditional branch. If after a certain maximum number of trials still no dependency is found that is not supposedly created by a store or a conditional branch instruction, the dependency is simply squashed.
3. If instruction x is a load operation, use the age of memory instances distribution to determine whether a store operation (before instruction x in the trace) accesses the same memory address.
4. If instruction x is a branch instruction, determine whether the branch and its target will be correctly predicted using the branch statistics.
5. If instruction x is a load operation, determine whether the load will get its data from L1 cache, L2 cache or main memory using the D-cache statistics.
6. Determine whether or not instruction x will cause an I-cache miss at the L1 or L2 level.

[1]Relative jumps, indirect jumps and returns do not have destination operands as well. However, we will not mention them for the remainder of this paper although we take this into account.

Trace-driven simulation. The last phase of the statictical simulation method is the trace-driven simulation of the synthetic trace which yields IPC (number of instructions retired per cycle). In order to model resource contention due to instructions along misspeculated control flow paths, we apply the following strategy when a branch instruction was marked as misspeculated by the synthetic trace generator: inject instructions into the simulator (using our synthetic trace generator) and mark them coming from a misspeculated execution path. When the misspeculated branch is executed, the instructions of the misspeculated path are squashed and instructions are fetched from the right control flow path.

Simulating statistical cache behaviour is done as follows. In case of an I-cache miss, the processor stops fetching new instructions as long as the I-cache is unresolved. In case of a D-cache miss, the load that causes an L1 or an L2 D-cache miss will get an L2 D-cache access time or a main memory access time assigned, respectively.

Simulation Speed. Our experiments showed that statistical simulation is a fast simulation technique due to the statistical nature of the technique. The standard deviation on the performance characteristics is less than 1% after simulating 1 million synthetically generated instructions.

5.2. Hybrid Analytical-Statistical Modeling

We make use of the information obtained in section 4 to transform the 'classical' statistical simulation technique into a methodology that bridges the gap between analytical and statistical modeling. We fitted the power law function to the measured conditional dependence probability. It is important to note that the measured conditional dependence probability is averaged over all register instances; no distinction is made per instruction type and per source register as is done in the 'classical' statistical simulation technique. This distribution fitting yields us two parameters α and β that fully characterize the age of register instances: $\alpha = P[X = 1]$ is the probability that an operation is dependent on its immediately preceeding operation and β is the slope of the conditional dependence probability in a

log-log diagram. As a result, the abstract workload model only contains a few parameters: the instruction mix (19 probabilities) and α and β to characterize the register traffic (and probabilities to characterize the branch and cache behaviour). This is in sharp contrast to the numerous probabilities included in a statistical profile of the 'classical' statistical simulation technique; e.g., in our environment 22 distributions are included each consisting of 512 probabilities. In addition, an analytical model offers possibilities to perform workload design space explorations, see section 7.

In the next two sections, we describe how this methodology can be used to do architecture design space explorations as well as workload design space explorations.

6. Architecture Design Space Exploration

6.1. Out-of-order architecture

To validate our performance modeling methodology, we assumed wide-issue out-of-order superscalar architectures. The four architectures considered are organized as follows: an instruction window *w* of 64, 128, 256 and 512 instructions; an issue width *i* of 4, 8, 12 and 16; 2, 4, 6 and 8 memory units; 3, 5, 8 and 10 non-memory units, respectively. The fetch bandwidth and the reorder bandwidth were chosen to be the same as the issue bandwidth. The latencies of the instruction types are: integer 1 cycle, load 3 cycles (this includes address calculation and L1 D-cache access), multiply 8 cycles, FP operation 4 cycles, single and double precision FP divide 18 and 31 cycles, respectively. All operations are fully pipelined, except for the divide. Dynamic memory disambiguation and selective re-execution to recover from misspeculated loads are modeled in our simulator.

The branch predictor is a hybrid predictor with a 4K meta predictor choosing between a 4K bimodal predictor and an 8-bit gshare that indexes into a 4K predictor [10]. The branch target buffer contains 512 sets and has an associativity of 4. The return address stack contains eight entries.

We considered two different memory subsystems in our evaluation section, a 'small' and a 'large' cache configuration. The 'small' configuration has an 8KB direct-mapped L1 I-cache, an 8KB direct-mapped L1 D-cache and a 64KB unified 2-way set-associative L2 cache. The 'large' configuration has a 32KB direct-mapped L1 I-cache, a 64KB 2-way set-associative L1 D-cache and a 256KB unified 4-way set-associative L2 cache. All caches have a block size of 32 bytes. A load that needs to access L2 cache takes 11 cycles (address calculation plus L2 cache access); a load that needs to access main memory takes 81 cycles. A L1 I-cache miss and a L2 I-cache miss take 10 and 80 clock cycles to resolve, respectively.

6.2. Performance Prediction Accuracy

To evaluate the absolute accuracy of the hybrid analytical-statistical modeling technique, we have measured the IPC prediction error, which is the error between the IPC by simulating a real trace and the IPC by simulating a synthetic trace generated from the profile corresponding to the real trace. We have plotted these IPC prediction errors (positive error means overestimation) for the various benchmarks in Figure 3 for the 'small' cache configuration. The 'large' cache configuration showed lower IPC prediction errors.

The IPC prediction errors are generally less than 20% even for wide out-of-order architectures which is quite acceptable for an early design stage modeling technique. Statistically modeling li and compress on the other hand, seems to be highly inaccurate for wide out-of-order processors. We believe that this is due to the fact that these benchmarks spend most of their time in tight loops which makes a statistical technique less powerful.

The IPC prediction errors for the hybrid analytical-statistical technique are comparable to the 'classical' statistical method. We can conclude that hybrid analytical-statistical modeling is nearly accurate as 'classical' statistical simulation.

6.3. Sensitivity Analyses

A modeling abstraction can also be evaluated by its relative accuracy, next to its absolute accuracy considered in the previous section. If the methodology estimates gradients in the design space in a reliable way, designers can make appropriate design decisions upon them. E.g., a low gradient indicates the computer architect not to further increase the resource since it will not further increase performance. Three examples of sensitivity analyses are shown in Figure 4: the sensitivity to load latency (performance degradation by increasing the load latency from 3 to 4 cycles on a 12-issue 256-entry window processor), issue width (performance gain by increasing the issue width from 8 to 16 in a 128-entry window processor) and window size (performance gain by increasing the window size from 128 to 512 in a 16-issue processor). These results are for processors with the 'large' cache configuration since the processors with the 'small' cache configuration showed low sensitivity to the above mentioned design parameters. The graphs in Figure 4 indicate that the 'classical' statistical and the hybrid analytical-statistical modeling technique are useful to make viable design decisions because the estimated gradients give a good indication of the real gradients.

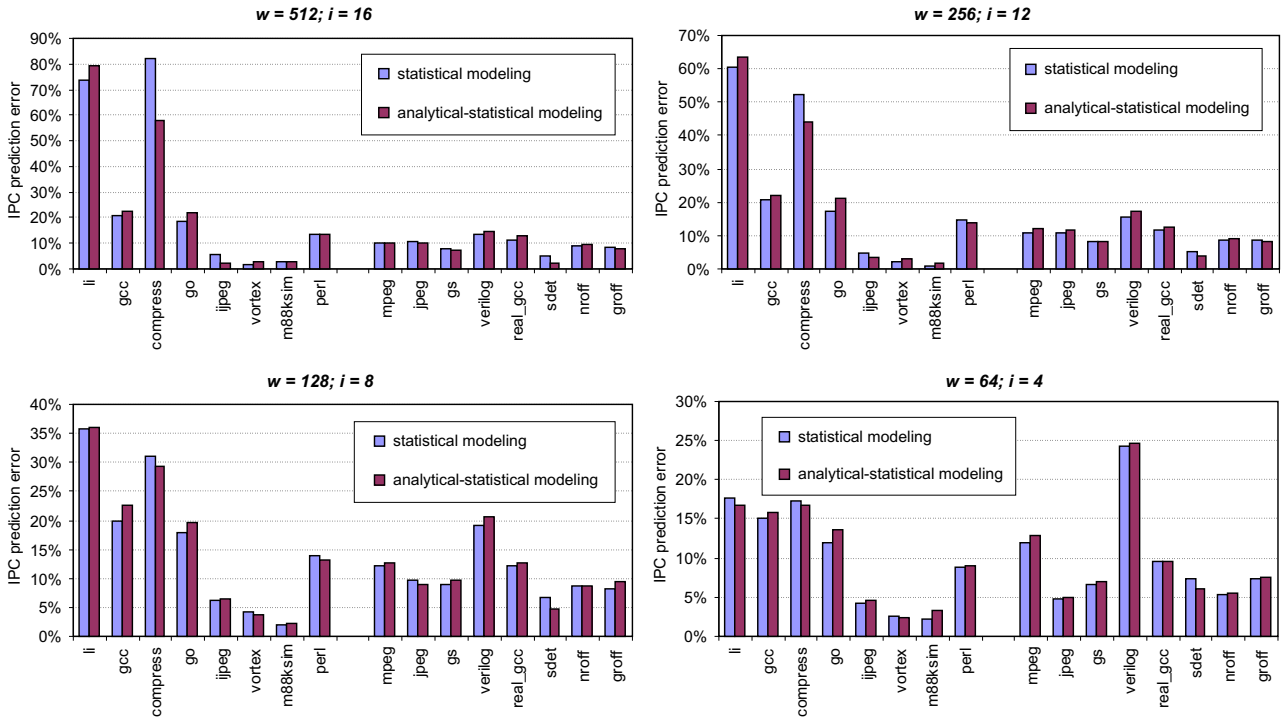

Figure 3. Performance prediction accuracy of the statistical and the hybrid analytical-statistical simulation methodology for various processor configurations (w is window size; i is issue width) with a 'small' cache configuration.

7. Workload Design Space Exploration

Theoretical workload modeling, of which hybrid analytical-statistical modeling is an example, is useful to provide insight in program behaviour and its implications to performance. For example, in section 4, the conditional dependence probability was fitted to the power law distribution with parameters α and β. Recall that α is the probability that an operation is dependent on its immediately preceeding operation and that β is the slope of the conditional dependence probability in a log-log diagram. From this observation we can conclude that a high β and a low α are an indication for the amount of ILP (instruction-level parallelism) available in a program. A high (low) β and low (high) α indicate many (few) dependencies over longer (shorter) distances and thus more (less) ILP. From a compiler point of view, β (α) can be increased (decreased) by techniques such as loop unrolling, scheduling, etc.

An important advantage of hybrid analytical-statistical modeling over 'classical' statistical modeling and performance evaluation using real benchmarks is that workload design space explorations can be done efficiently. Theoretical workload models allow us to explore the *entire* workload design space by specifying a limited number of parameters. Real benchmark suites on the other hand, only provide a sample from the entire workload space and although it is possible to explore the workload space by 'classical' statistical modeling, it is highly impractical due to the numerous number of probabilities that need to be specified. An example of a workload design space exploration is shown in Figure 5: IPC as a function of α and β for a 16-issue 512-entry window processor with perfect caches and a 97% branch prediction accuracy. Performance is more sensitive to α and β for low values of β and high values of α, respectively. For low (high) values of α (β), performance saturates for higher (lower) values of β (α) which means that the available ILP remains unexploited due to branch mispredictions. It is also interesting to note that there exist points in the design space where different values for α and β result in the same performance, e.g., $\alpha = 0.2$ and $\beta = 0.7$ vs. $\alpha = 0.25$ and $\beta = 0.8$.

Theoretical workload modeling can also be used to investigate the interaction between program characteristics that are hard to vary in real programs: e.g., the interaction between the age of register instances and the D-cache miss rate, as is shown in Figure 6. This graph clearly shows that the latency of load operations that miss in the data cache can be effectively hidden by a longer age of register instances.

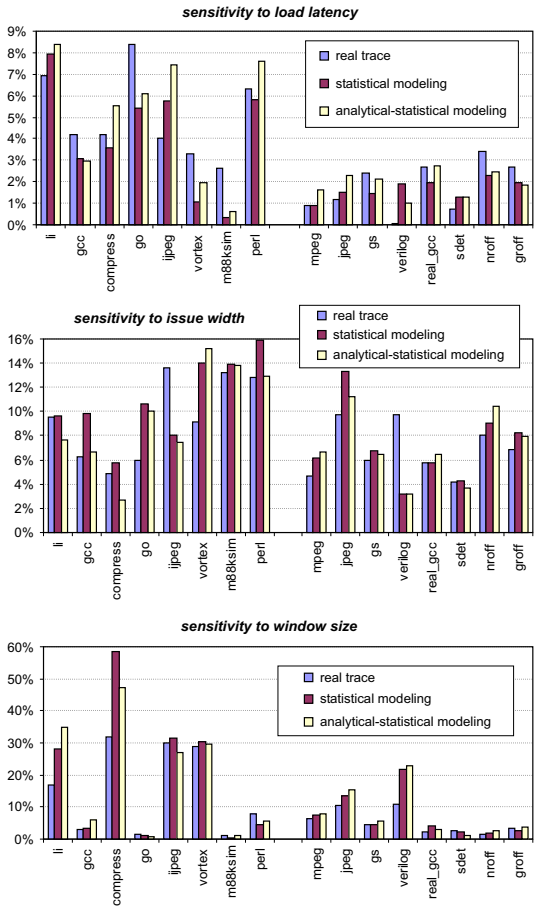

Figure 4. Sensitivity analyses: performance degradation when increasing the load latency from 3 cycles to 4 cycles (upper graph); performance gain when increasing the issue width from 8 to 16 in a 128-entry window processor (graph in the middle); performance gain when increasing the window size from 128 to 512 in a 16-issue processor (lower graph).

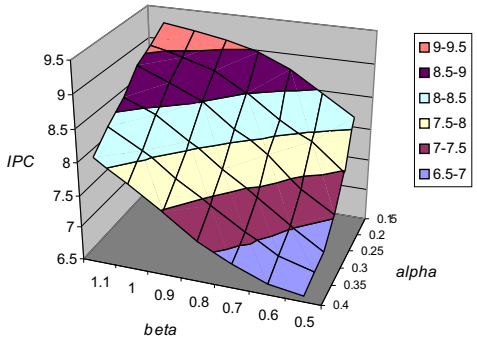

Figure 5. Workload design space: IPC as function of α and β for a 16-issue, 512-entry window processor with perfect caches and a 97% branch prediction accuracy.

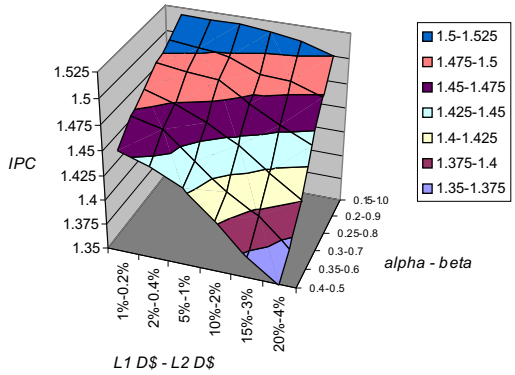

Figure 6. Workload design space: IPC as function of α, β, L1 and L2 D-cache miss rate for a 16-issue, 512-entry window processor with a 2% L1 I-cache miss rate, a 0.5% L2 I-cache miss rate and a 97% branch prediction accuracy.

We also observe that the performance improvement due to enlarging the age of register instances will be modest for applications with low D-cache miss rates.

Similar workload design space explorations were done in previous work but all of them used a workload model that does not resemble real program characteristics which may lead to incorrect conclusions. Dubey, Adams and Flynn [3] assumed a constant conditional independence probability p_x; Kamin, Adams and Dubey [9] assumed an exponential approximation of the conditional independence probability which is inaccurate for real workloads, see section 4. Oskin, Chong and Farrens [13] used the average dependency distance between instructions to impose register dependencies which is unrealistic as well, see section 3.

8. Related Work

Dubey, Adams and Flynn [3] and the continuation of that work done by Kamin, Adams and Dubey [9] proposed an analytical performance model on the basis of two parameters that can be extracted from a program trace. Noonburg and Shen [11] presented an analytical model that uses distributions of parallelism measured from real program traces, without actually modeling inter-operation dependen-

cies. Noonburg and Shen continued their work in [12] where they presented a framework that models the execution of a program on a particular architecture as a Markov chain, in which the state space is determined by the microarchitecture and in which the transition probabilities are determined by the program. Sorin *et al.* [14] proposed an analytical model for evaluating shared-memory systems with ILP processors.

Hsieh and Pedram [7] present a technique to estimate performance and power dissipation of a microprocessor by first measuring a characteristic profile of a program execution, and by subsequently synthesizing a new, fully functional program that matches the extracted characteristic profile. The characteristic profile includes the instruction mix, branch prediction accuracy, cache miss rate, pipeline stall rate and IPC. The major disadvantage of this approach is that all characteristics are microarchitecture-dependent. Consequently, this approach cannot be used for architecture nor workload design space explorations.

Carl and Smith [2] proposed an approach, where a synthetic instruction trace was generated based on a statistical profile and fed into a trace-driven simulator. Oskin, Chong and Farrens [13] used a similar approach in HLS to perform design space explorations of microprocessors in an efficient way. Our previous work focused on guaranteeing syntactical correctness of the synthetic trace—i.e., stores and conditional branches should not have a destination operand—while preserving the representativeness of the generated trace [5]; and on increasing the accuracy of statistical simulation by modeling bursty cache miss behaviour [4].

9. Conclusion

Microprocessor design time is getting impractically long due to long simulation runs that need to be done to evaluate various processor configurations for various workloads. In this paper, we have presented an early design stage methodology that bridges the gap between analytical and statistical modeling. This method can be used to efficiently cull huge design spaces to identify a region of interest that needs to be further analyzed through more accurate simulations using real benchmarks. The hybrid analytical-statistical method proposed in this paper is based on the observation that register traffic characteristics, such as the age of register instances, exhibit power law properties. This observation motivates our approach to fit a power law distribution to the conditional dependence probability which is more accurate than previously proposed approximations. This allows us to present an analytical workload method that fully characterizes a workload based on just a few parameters. In this paper, we have also demonstrated the applicability and the accuracy of the hybrid analytical-statistical modeling methodology to explore architecture as well as workload design spaces which would have been difficult, if not impossible, using other methodologies.

Acknowledgments

Lieven Eeckhout is supported by a grant from the Flemish Institute for the Promotion of the Scientific-Technological Research in the Industry (IWT). The authors would like to thank the reviewers for their valuable comments.

References

[1] P. Bose and T. M. Conte. Performance analysis and its impact on design. *IEEE Computer*, 31(5):41–49, May 1998.

[2] R. Carl and J. E. Smith. Modeling superscalar processors via statistical simulation. In *Workshop on Performance Analysis and its Impact on Design*, June 1998.

[3] P. K. Dubey, G. B. Adams III, and M. J. Flynn. Instruction window size trade-offs and characterization of program parallelism. *IEEE Transactions on Computers*, 43(4):431–442, Apr. 1994.

[4] L. Eeckhout and K. D. Bosschere. Increasing the accuracy of statistical simulation for modeling superscalar processors. In *The 20th IEEE International Performance, Computing and Communications Conference (IPCCC 2001)*, pages 196–204, Apr. 2001.

[5] L. Eeckhout, K. D. Bosschere, and H. Neefs. Performance analysis through synthetic trace generation. In *The IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2000)*, pages 1–6, Apr. 2000.

[6] M. Franklin and G. S. Sohi. Register traffic analysis for streamlining inter-operation communication in fine-grain parallel processors. In *Proceedings of the 22nd Annual International Symposium on Microarchitecture (MICRO-22)*, pages 236–245, Dec. 1992.

[7] C. Hsieh and M. Pedram. Micro-processor power estimation using profile-driven program synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(11):1080–1089, Nov. 1998.

[8] L. K. John, P. Vasudevan, and J. Sabarinathan. Workload characterization: Motivation, goals and methodology. In L. K. John and A. M. G. Maynard, editors, *Workload Characterization: Methodology and Case Studies*. IEEE Computer Society, 1999.

[9] R. A. Kamin III, G. B. Adams III, and P. K. Dubey. Dynamic trace analysis for analytic modeling of superscalar performance. *Performance Evaluation*, 19(2-3):259–276, Mar. 1994.

[10] S. McFarling. Combining branch predictors. Technical Report WRL TN-36, Digital Western Research Laboratory, June 1993.

[11] D. B. Noonburg and J. P. Shen. Theoretical modeling of superscalar processor performance. In *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO-27)*, pages 52–62, Nov. 1994.

	li	gcc	compress	go	ijpeg	vortex	m88ksim	perl
dyn. instr. count (M)	226	182	217	200	170	200	200	200
stat. instr. count (90%)	580	24,084	336	6,744	1,150	2,544	1,097	1,546
α	0.241	0.234	0.236	0.206	0.209	0.199	0.271	0.240
β	0.702	0.731	0.813	0.682	0.765	0.598	0.922	0.632
% conditional branches	11.22%	12.70%	7.17%	10.10%	8.69%	9.33%	7.67%	10.95%
prediction accuracy	93.99%	91.49%	89.17%	79.01%	90.91%	99.03%	95.87%	96.21%
% loads	34.36%	30.18%	31.02%	37.19%	24.70%	28.38%	29.12%	29.84%
% stores	13.83%	10.98%	4.80%	7.32%	5.21%	14.62%	14.09%	14.60%
8KB DM L1 I-cache; 8KB DM L1 D-cache; 64KB 2WSA unified L2 cache								
L1 I-cache miss rate	2.17%	3.87%	<0.01%	3.53%	0.03%	5.68%	3.67%	4.30%
L2 I-cache miss rate	0.03%	0.95%	<0.01%	0.65%	<0.01%	0.50%	1.95%	0.38%
L1 D-cache miss rate	4.42%	7.10%	3.88%	8.32%	3.42%	10.63%	6.66%	16.50%
L2 D-cache miss rate	2.67%	2.27%	2.42%	0.73%	0.39%	1.33%	0.25%	1.43%
32KB DM L1 I-cache; 64KB 2WSA L1 D-cache; 256KB 4WSA unified L2 cache								
L1 I-cache miss rate	0.03%	1.67%	<0.01%	1.38%	<0.01%	1.59%	3.85%	1.02%
L2 I-cache miss rate	<0.01%	0.19%	<0.01%	0.01%	<0.01%	0.03%	<0.01%	<0.01%
L1 D-cache miss rate	2.43%	1.12%	1.45%	0.06%	0.02%	0.27%	0.01%	0.54%
L2 D-cache miss rate	0.20%	0.28%	0.97%	0.03%	0.36%	0.46%	0.02%	0.02%

Table 1. Characteristics of the SPECint95 traces.

	mpeg	jpeg	gs	verilog	real_gcc	sdet	nroff	groff
dyn. instr. count (M)	99	97	106	47	110	38	110	97
stat. instr. count (90%)	7,357	1,740	7,895	5,631	21,940	7,804	2,014	4,346
α	0.268	0.248	0.259	0.233	0.270	0.259	0.328	0.269
β	0.825	0.749	0.781	0.714	0.791	0.835	0.975	0.751
% conditional branches	9.63%	16.31%	14.45%	13.68%	15.01%	10.68%	20.81%	12.89%
prediction accuracy	95.27%	98.73%	97.15%	97.04%	93.82%	95.35%	98.23%	97.24%
% loads	25.03%	16.93%	22.66%	27.74%	24.26%	24.81%	21.54%	26.63%
% stores	17.77%	7.42%	14.11%	20.35%	13.92%	16.38%	10.23%	15.00%
8KB DM L1 I-cache; 8KB DM L1 D-cache; 64KB 2WSA unified L2 cache								
L1 I-cache miss rate	2.62%	1.50%	4.50%	4.31%	3.98%	3.96%	3.16%	6.09%
L2 I-cache miss rate	2.06%	1.01%	1.30%	1.20%	1.32%	2.69%	0.48%	1.33%
L1 D-cache miss rate	7.35%	4.27%	6.73%	4.56%	5.35%	5.57%	6.83%	4.86%
L2 D-cache miss rate	4.65%	2.99%	2.57%	7.93%	2.38%	4.73%	1.14%	1.78%
32KB DM L1 I-cache; 64KB 2WSA L1 D-cache; 256KB 4WSA unified L2 cache								
L1 I-cache miss rate	2.04%	1.28%	2.33%	1.90%	2.14%	3.27%	1.83%	3.13%
L2 I-cache miss rate	0.43%	0.10%	0.21%	0.26%	0.31%	0.49%	0.05%	0.10%
L1 D-cache miss rate	1.85%	0.94%	0.59%	3.62%	0.71%	1.32%	0.48%	0.46%
L2 D-cache miss rate	1.16%	0.72%	0.83%	3.53%	0.62%	1.61%	0.23%	0.37%

Table 2. Characteristics of the IBS traces.

[12] D. B. Noonburg and J. P. Shen. A framework for statistical modeling of superscalar processor performance. In *Proceedings of the third International Symposium on High-Performance Computer Architecture (HPCA-3)*, pages 298–309, Feb. 1997.

[13] M. Oskin, F. T. Chong, and M. Farrens. HLS: Combining statistical and symbolic simulation to guide microprocessor design. In *Proceedings of the 27th Annual International Symposium on Computer Architecture (ISCA-27)*, pages 71–82, June 2000.

[14] D. J. Sorin, V. S. Pai, S. V. Adve, M. K. Vernon, and D. A. Wood. Analytic evaluation of shared-memory systems with ILP processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture (ISCA-25)*, pages 380–391, June 1998.

[15] R. Uhlig, D. Nagle, T. Mudge, S. Sechrest, and J. Emer. Instruction fetching: Coping with code bloat. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture (ISCA-22)*, pages 345–356, June 1995.